MUSE: Mining and Understanding Software Enclaves

Reasoning with Big Code

Suresh Jagannathan, I2O

March 7, 2014

MUSE Proposers' Day Agenda

0900 – 1000	Check-In/Registration	
1000 – 1050	Welcome and Overview of MUSE Program	Dr. Suresh Jagannathan, DARPA
1050 – 1105	Contracts	Mr. Mark Jones, DARPA
1105 – 1115	Security	Mr. Wayne Warner, DARPA
1115 – 1245	LUNCH BREAK	
1245 – 1345	Individual Company Presentations	
1345 – 1445	Government Response to Questions	Dr. Suresh Jagannathan, DARPA

Logistics

- DARPA-BAA 14-22
  - Posted on FedBizOpps website (http://www.fedbizopps.gov) and Grants.gov website (http://www.grants.gov)
  - Posting Date: February 25, 2014
  - Proposal Due Date: April 15, 2014 at Noon ET

- Procedure for Questions/Answer
  - Questions can be submitted until 1145 this morning to MUSE@darpa.mil
  - Questions will be answered during Q&A session in the afternoon

- Program website http://www.darpa.mil/Our_Work/I2O/Programs/Mining_and_Understanding_Software_Enclaves_%28MUSE%29.aspx
  - Copy of presentations
  - Video recording
  - Frequently Asked Questions (FAQs)

A myriad set of techniques for software validation

Testing

PLT Redex: DSL for specifying, debugging, and testing operational semantics

Quickcheck: Specification-driven formulation of properties that can be checked using random testing

Csmith: Random generator of C programs that conform to C99 standard for stress-testing compilers, analyses, etc.

CUTE: Unit-testing of C programs with pointer arguments by combining symbolic and concrete executions

Korat: Constraint-based generation of complex test inputs for Java programs, focusing on data structures and invariants

Symbolic Execution

KLEE: Symbolic execution engine to generate high-coverage test cases

S2E: Scalable path-sensitive platform

Static Program Analysis

CFA: Whole-program control-flow analysis that computes the set of procedures that can be invoked at a call-site

ASTREE: Abstract interpretation of real-time embedded software designed to prove absence of runtime errors by overapproximation of program behavior

TVLA: Flow-sensitive shape analysis of dynamically-allocated imperative data structures

Bddbdddb: Context- and field-sensitive analysis applied to Java that translates analysis rules expressed in Datalog to BDD representation

Saturn: Scalable and modular summary-driven bit-level constraint-based analysis framework

Coverity: Unsound scalable analyses used to check correctness of C, C++, and Java programs.

Dynamic Program Analysis

Contracts : Assertions checked at runtime with blame

Daikon: Likely pre- and post-condition invariant detection over propositional terms, based on program instrumentation and

Valgrind: Instrument binary programs to track memory access violations and data races using dynamic recompilation

Fasttrack: Lightweight data race detector that uses vector clocks and a dynamically constructed happens-before relation

Model Checking

CVC, SLAM, Blast, Spin, Java PathFinder

CHESS: Bounded model-checking for unit-testing of shared-memory concurrent programs

TLA: Temporal logic of actions for specifying and checking concurrent systems

Logics and Types

Jstor, Space Invader, Smallfoot: Separation-logic based tools for verifying expressive shape properties of dynamic data structures and heaps

ESC: Extended static checking that combines type checking with theorem proving

Coq. Agda, Isabelle, ACL2, NuPRL: Mechanized proof assistants

Ynot: Hoare Type Theory

Rely-Guarantee Reasoning: Modular verification of shared-memory concurrency

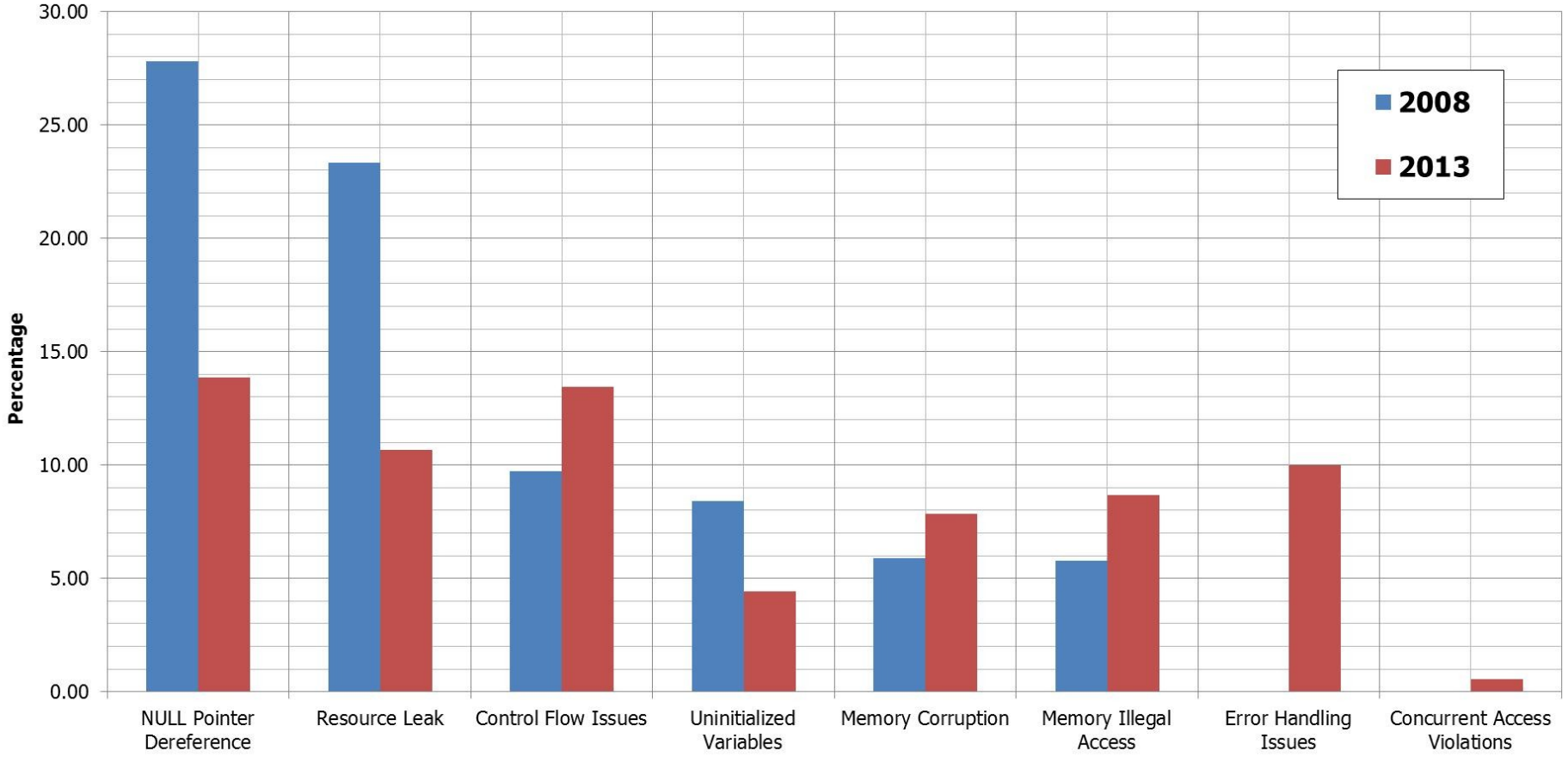
Liquid Type Inference: Discovery of expressive refinement properties in Haskell, ML, and C

Hybrid Type Checking and Soft Typing

Session Types: type systems for expressing communication protocols

But, bugs still remain difficult to prevent, catch, and repair

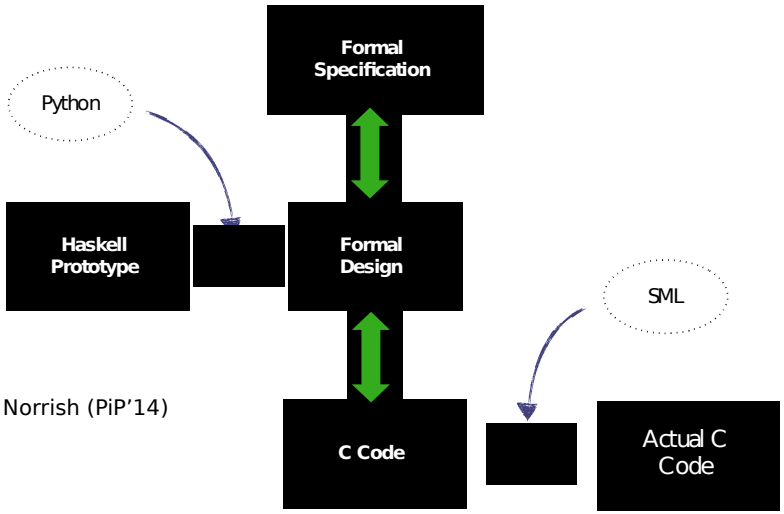
- Average defect density is *2.3x* greater in 2013 (.69) than in 2008 (.3)
  - Defect Density = # of bugs/KLoC



Source: Coverity, Open Source Report, 2013

From Specifications to Implementations

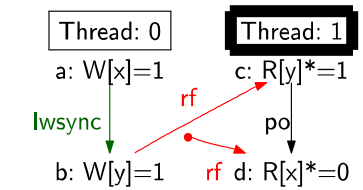
Specifications may be complex



Source: Norrish (PiP'14)

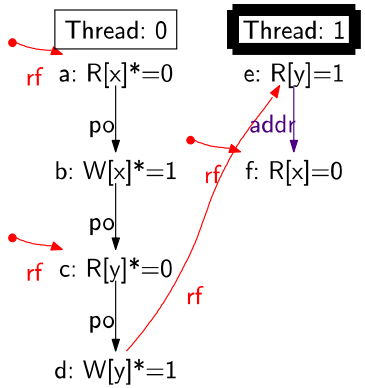
Sel4 architecture: Specification defined across multiple abstraction layers which include various untrusted translation phases.

Specifications may be intensional



Test MP+lwsync+porr: Forbidden

Source: Sarkar et. al (PLDI'12



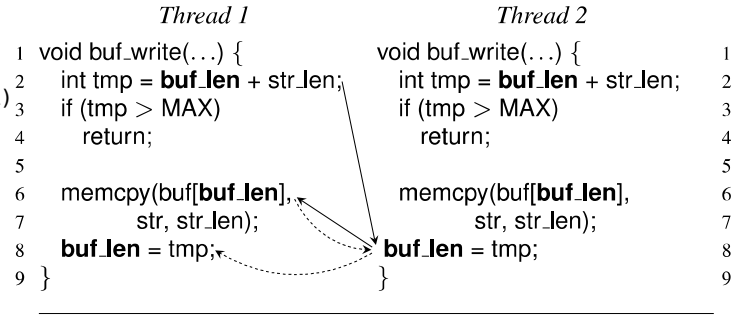
Test MP+poaa+addr: Allowed

Power relaxed-memory model: Allowed program behaviors depend on visibility and ordering guarantees of underlying processor.

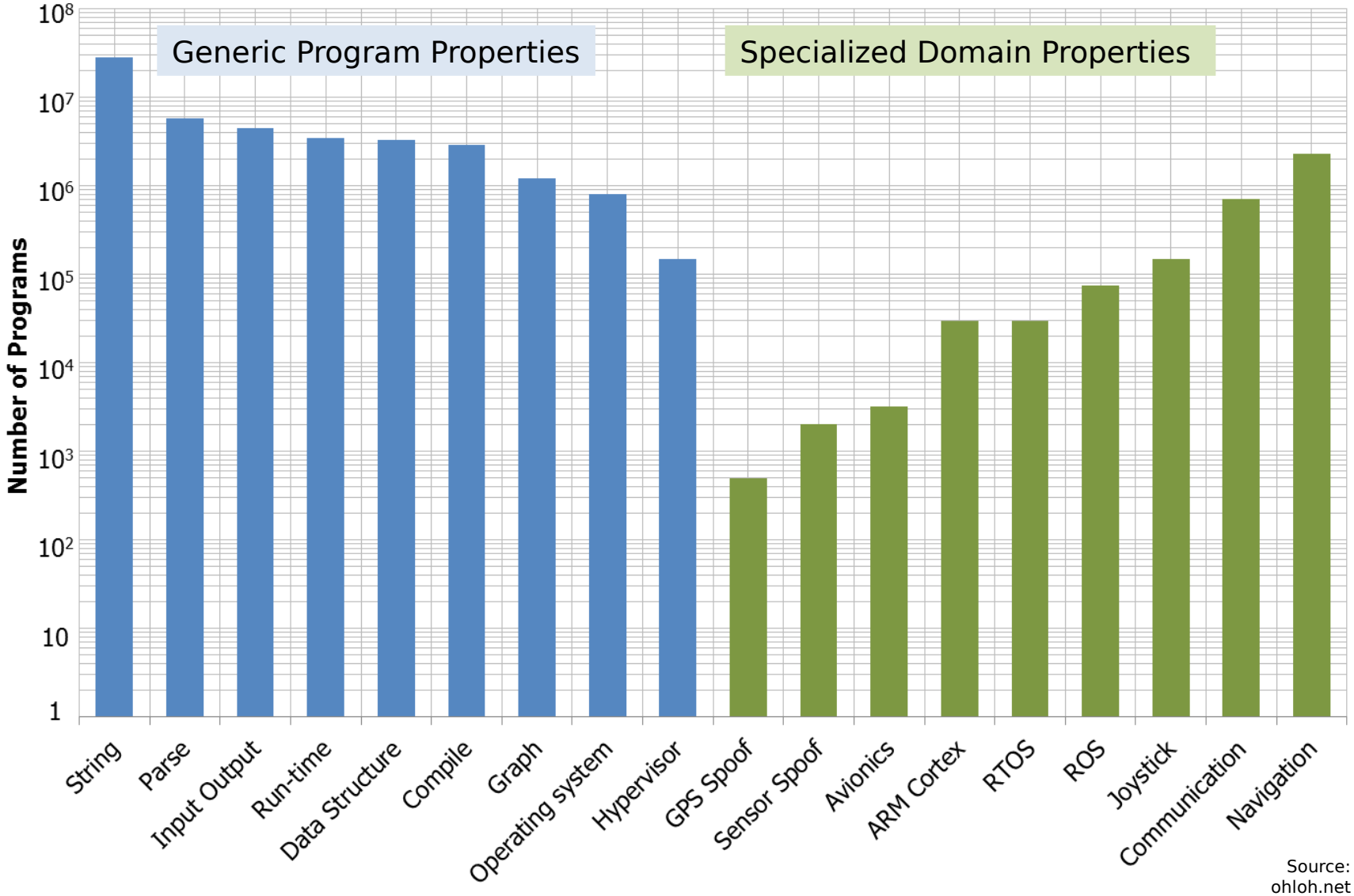
Specifications may express non-local (global) and inter-related invariants

Source: Lin et. al (PLDI'11)

Afix: Two atomicity violations, but treating the fixes independently can lead to deadlock



Knowledge extraction and redundancy in software

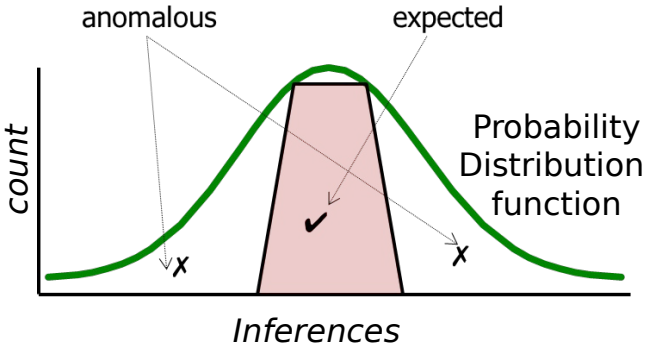


Source: ohloh.net

Exploiting redundancy: Big Data for Software Analytics

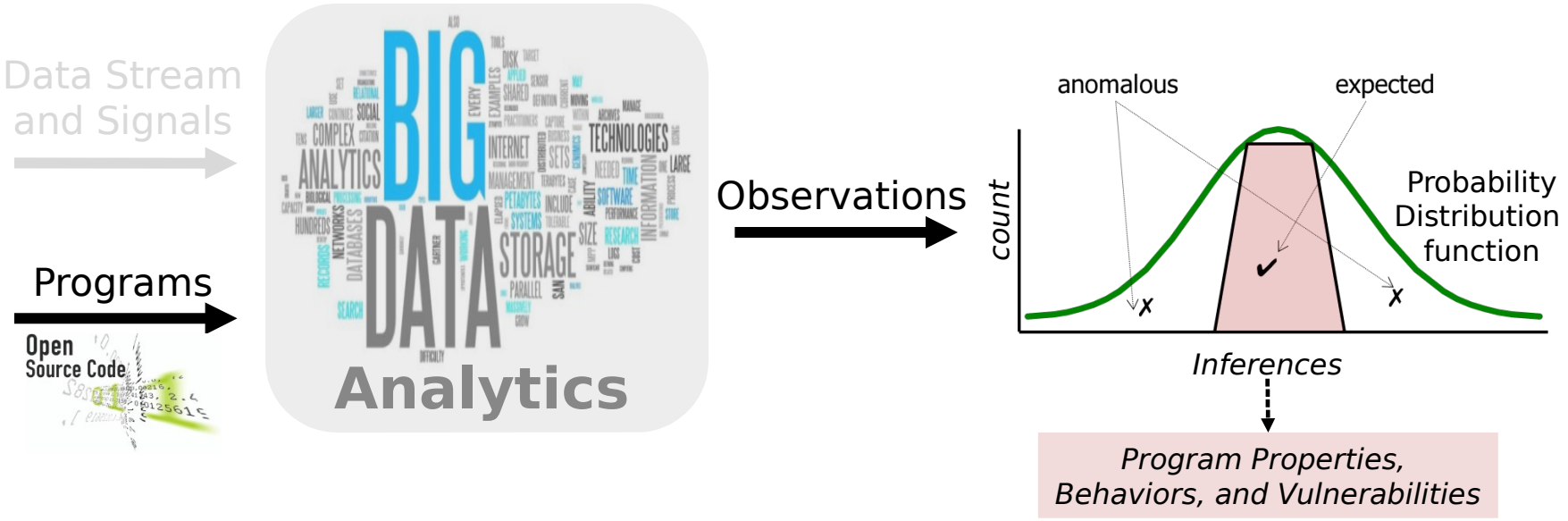
Data Stream and Signals

Observations



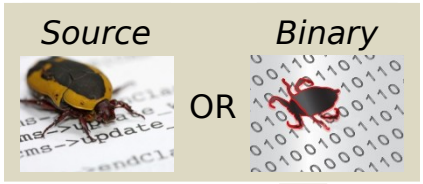
Big Data for Software Analytics

Apply principles of *Big Data Analytics* to a large corpus of *Open-Source Multi-Lingual Software*

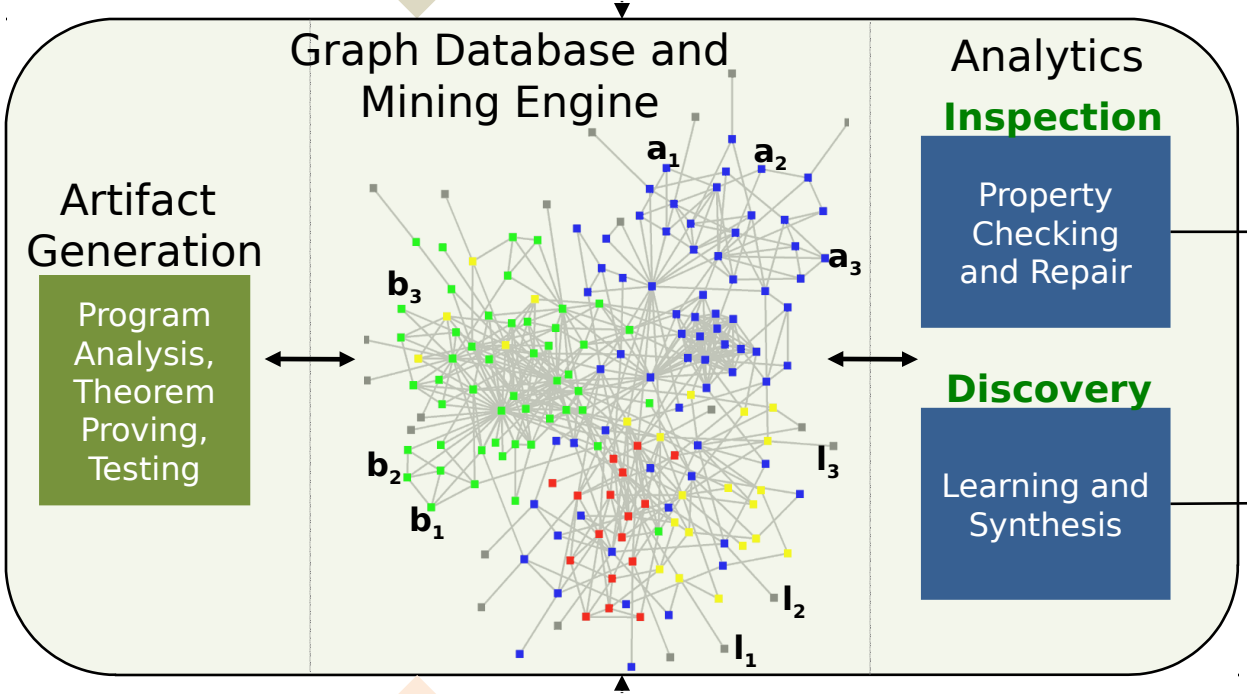


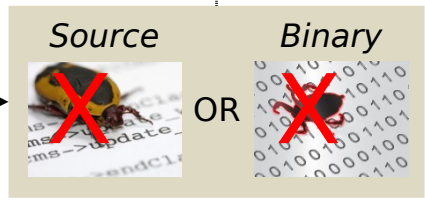
- Treat programs (more precisely, semantic objects extracted from programs) as data
- Observations and inferences applied to program properties

Design



Paradigm shift: replace costly and laborious test/debug/validate cycle with "always on" program analysis, mining, inspection, & discovery

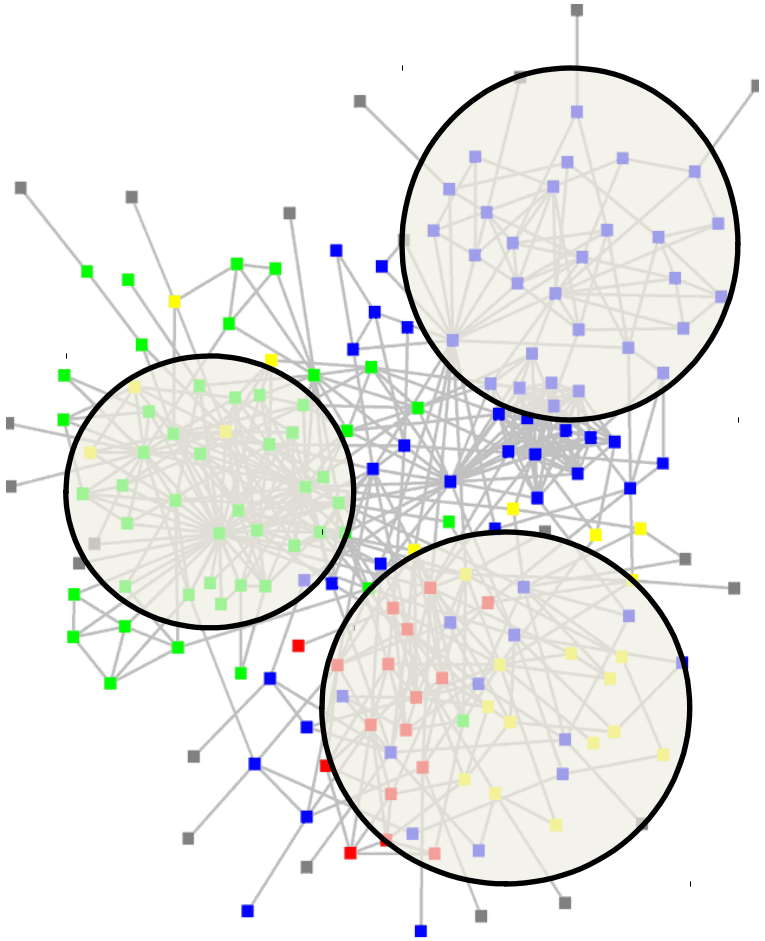




Program that satisfies X:
$\underline{f}(a_1) \circ \underline{g}(b_2) \circ \underline{h}(l_3)$

Query: "Synthesize a program that does X"

Enclaves



Redundancies in the corpus exposed as dense components (*enclaves*) in the mined network

- *Nodes* represent properties facts, claims, and evidence
- *Edges* connect related properties

Anomalous properties have small number of connections

Likely invariants have large number of connections

Challenges: Big Code Front-End

Collector!

Diverse, Representative,!
High-fidelity corpus!

Classifier!

Ontological Structure!

Environment and platform dependencies,!
Models (memory, execution, …)!

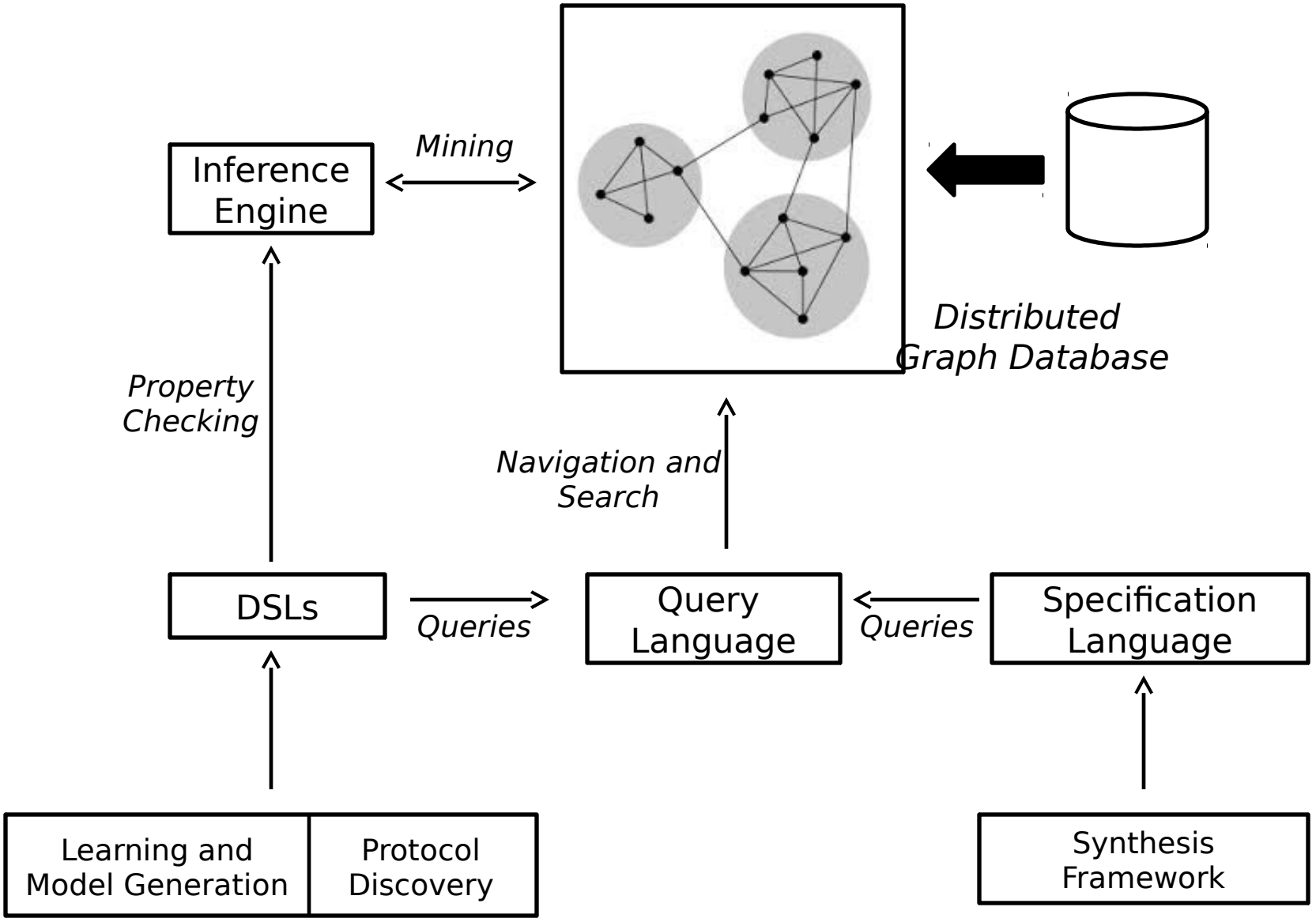
Types and Proofs!
Binary decompilation!
Static and dynamic analyses!
Theorem proving!
Tests and runtime verification!
Executable Specifications!
Model Checking!
Abstract Interpretation!
Contracts and assertions!
Documentation extraction!

Program! Analyses!

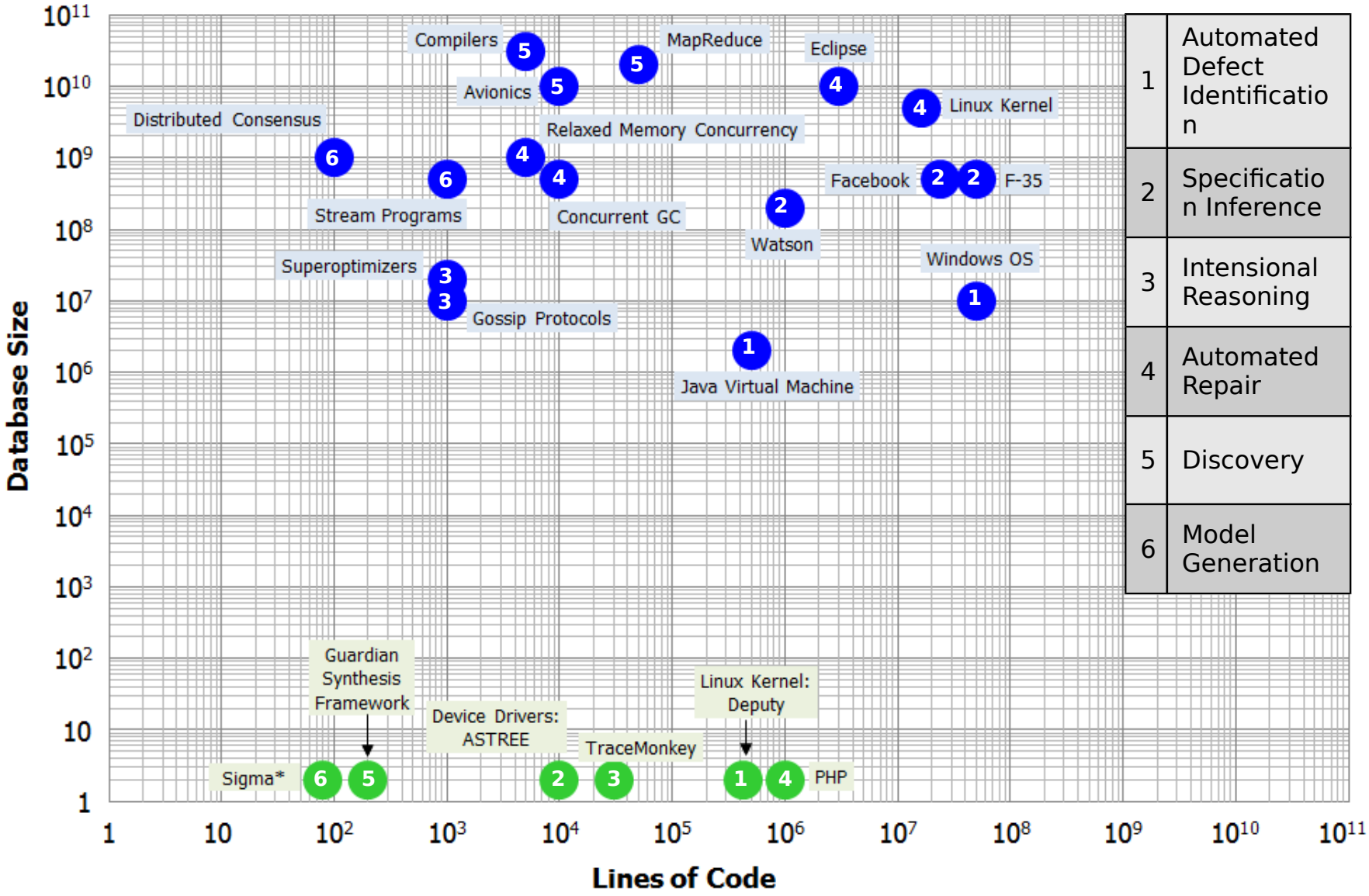
Canonical and persistent representation of analysis outputs

Database ! construction!

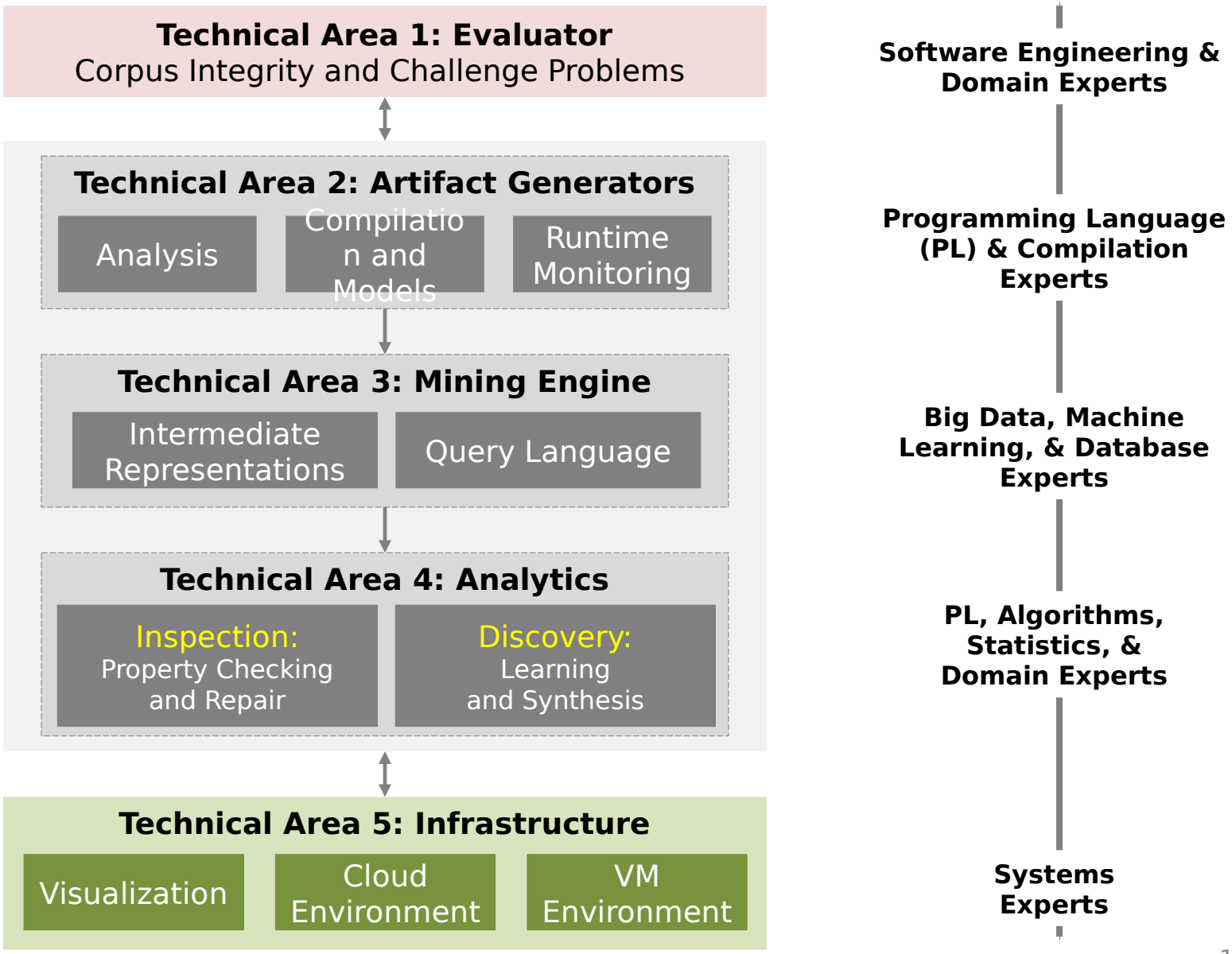
Challenges: Big Code Back-End



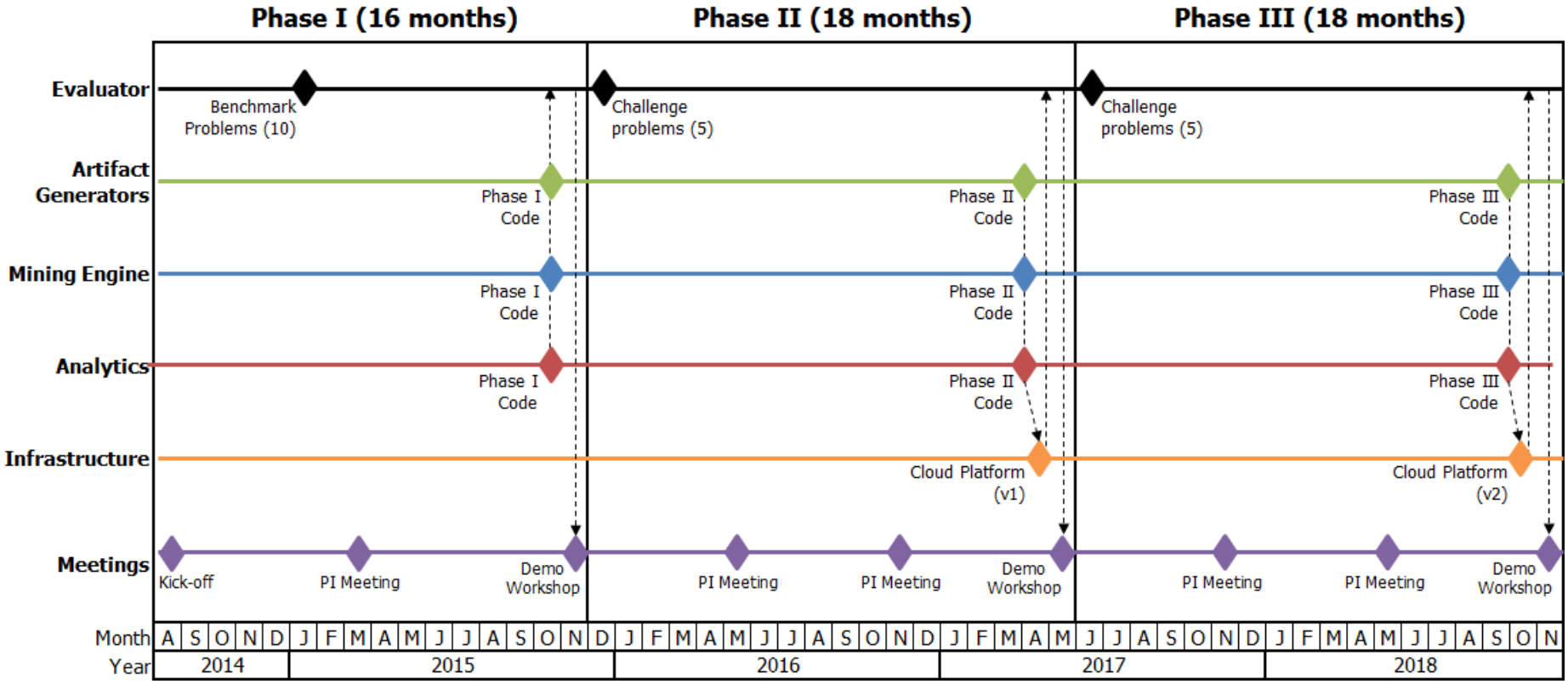
If this technology is wildly successful…



Program Structure



Schedule, Products, & Transition Plan



Products

- Mining engine
- Large semantic corpus
- Repair and discovery analytics
- Multi-lingual support for program analysis
- Supporting community

Programmatics

- Three planned phases: Phase I (16 months); Phases II & III (18 months each)
- Five Technical Areas (TAs)
  - TA1 - Evaluator
  - TA2 - Artifact Generators
  - TA3 - Mining Engine
  - TA4 - Analytics
  - TA5 - Infrastructure
- Anticipate one award for TA1 and TA5 and multiple awards in each of TA2-4
  - If selected for TA1 and TA5, cannot be selected for any portion of the other TAs
- Performers in TA2-4 will be grouped into one or more design teams
  - Each team led by a TA4 performer
  - Each team will have one TA3 performer and one or more TA2 performers
  - Each team will produce a working end-to-end Analytic and Artifact System (AAS)
  - Teams will not be competitively evaluated; no anticipated down-selection
- Strong interaction among all performers is critical to program success
  - Associate Contractor Agreement (ACA)

Technical Area 1 – Evaluator

- Develop benchmarks and Challenge Problems (CPs) for TA2-4 performers
  - 10 benchmarks due 6 months after kickoff
  - At least 5 CPs due 1 month after start of Phases II and III
    - Grow in complexity throughout program
- Develop rich corpus of open source and open binary software for TA2 and TA4 performers
- Evaluate performance of each AAS on each benchmark and CP
  - Quality of the solution
  - Run-time performance is secondary
  - Present results at Demo Workshops
- Lead Demo Workshops (end of each phase)
  - Focus on strengths and weaknesses of existing implementation approaches w.r.t. tackling proposed problems
  - Provide retrospective analysis on the effectiveness of strategies on the problems presented at the beginning of the phase

- Consult with the Government Team and TA2-4 performers in selecting benchmarks and CPs
- Regarding the CPs, make available to TA2-4 performers any reference material to measure the effectiveness of their approaches
- Regarding the corpus, make available to TA2-4 performers any test scripts or other information
- Work with performers from all TAs to identify critical research challenges and issues

Technical Area 2 – Artifact Generators

- Design and build “front-end” of a AAS
  - Populate the database with facts, evidence, conjectures, and inferences about the programs (and the properties they satisfy) comprising the corpus
    - Program analyses, both static and dynamic, tests, expressive type checking and inference tools, automated verification tools (e.g., constraint, SMT, or SAT solvers), model checking, abstract interpretation, shape analysis, effect analysis, concurrency analysis, mechanized proof assistants, etc.
  - Provide support for binary analysis and decompilation for binary code in the corpus
  - Provide a sufficiently detailed understanding of input programs to allow effective mining and inference generation by backend analytic frameworks

- Participate in one AAS design team, led by a TA4 performer
- Work with both TA3 and TA4 performers to incorporate novel analyses, algorithms, and representations to support the models and inference strategies adopted by TA4 performers
- Use the benchmarks and CPs developed by the TA1 performers and the Team CP to evaluate and drive research
- Work with performers from all TAs to identify critical research challenges and issues

Technical Area 3 – Mining Engine

- Build and maintain a persistent graph database store
  - Output of analyses developed in TA2
  - Input to analytics developed in TA4
  - Provide efficient and scalable access to TA2 and TA4 performers to populate, refine, mine, and navigate the database
  - Develop new APIs and interfaces, query and specification languages, and representation schemes that accommodate the varied analyses and analytics developed by TA2 and TA4 performers
- Not required to implement the graph database from scratch; allowed to use and customize open-source database engines (e.g., Titan or Neo4j)

- Participate in one or more AAS design teams, led by a TA4 performer
- Work with both TA2 and TA4 performer to tailor the structure of the interfaces they provide to best reflect the needs of the particular analyses developed in TA2 and analytics developed in TA4
- Use the CPs developed by the TA1 performers and the Team CP to evaluate and drive research
- Work with performers from all TAs to identify critical research challenges and issues

Technical Area 4 – Analytics

- Design and build the “back-end” of a AAS
  - Generate global inferences from the data captured within the database
  - Apply these inferences on problems related to property checking, repair, learning, and synthesis, among others
  - Apply deep inspection of the database to establish relations among components derived from a multitude of programs
    - association rule mining
    - frequent itemset mining
    - clustering techniques
    - classification strategies
    - online learning algorithms like L*
    - etc.
- Demonstrate the AAS applied to the Team CP and benchmarks at the Demo Workshop

- Lead a AAS design team
  - One TA3 performer
  - One or more TA2 performers
  - Develop Team CP
- Propose one or more benchmarks and CPs to the TA1 performer
- Deliver AAS to TA1 performer one month before each Demo Workshop
- Serve as primary point of contact for technical support to the TA1 performer during the evaluation of the AAS on program-wide as well as team specific benchmarks and CPs
- Work with performers from all TAs to identify critical research

Technical Area 5 – Infrastructure

- Overall integrator for the program
  - Provide facilities to house the corpus, and the tools, implementations, and systems developed by the AAS teams
  - Phase I: handle storage needs for the corpus
  - Phases II and III: provide a cloud infrastructure that the TA1 performer and AAS teams will use to manage development and data warehousing needs
- Provide a virtualization environment that allows corpus programs to be executed within the appropriate environment
- Produce visualization tools that AAS teams can use to understand the structure of the graph produced by their analysis and analytic techniques

- Work closely with performers from all TAs in Phases II and III to ensure code, corpora, documentation, and environments are properly housed and accessible via the cloud infrastructure
- Work with performers from all TAs to identify critical research challenges and issues